(19)日本国特許庁（ＪＰ）　(12)特　許　公　報（Ｂ２）　(11)特許出願公告番号

特公平7－86836

(24)(44)公告日　平成７年(1995)９月20日

(51)Int.Cl.⁶	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/45				
12/02	５８０　Ｈ	7608－5B		
		7737－5B	Ｇ０６Ｆ　9/44　３２２　Ｈ	

発明の数１(全 12 頁)

(21)出願番号　特願昭61－274428

(22)出願日　昭和61年(1986)11月18日

(65)公開番号　特開昭63－127340

(43)公開日　昭和63年(1988)５月31日

(71)出願人　999999999
株式会社日立製作所
東京都千代田区神田駿河台４丁目６番地

(72)発明者　梅野　佳津子
神奈川県横浜市戸塚区戸塚町5030番地　株式会社日立製作所ソフトウエア工場内

(74)代理人　弁理士　磯村　雅俊

審査官　吉岡　浩

(56)参考文献　特開　昭52－79737（ＪＰ，Ａ）

(54)【発明の名称】　多次元配列のメモリ割付け方式

1

【特許請求の範囲】

【請求項１】複数のバンクから構成されるメモリに多次元配列の要素を順次静的に割付けるコンパイラにおいて、各該バンクへの割付け方向に沿った要素の数がバンク数の整数倍である場合に、該方向の要素に対して仮想的要素を追加し、該方向と直交する方向に沿った要素を複数のバンクに分散して割付けることを特徴とする多次元配列のメモリ割付け方式。

【発明の詳細な説明】

〔産業上の利用分野〕

本発明は原始プログラムから目的プログラムを生成するコンパイラに係わり、特に多次元配列を静的にメモリに割付けるときデータアクセス性能が低下しないように割付け可能な、多次元配列のメモリ割付け方式に関する。

〔従来の技術〕

2

FORTRANNT等のコンパイラにおける配列のメモリへの割付けは、配列毎に作られる配列記述テーブルに従って行われる。このテーブルには、配列の次元の数や、配列長，要素数，各次元の寸法等の値を設定するが、従来のコンパイラは、例えば、小田一博著「コンパイラ入門文法から設計・製作まで」（日本評論社（1971），pp218～223）に記載されている如く、利用者がプログラムで宣言した通りの大きさを前記配列記述テーブルに設定しており、それがバンクコンフリクトやバッファの頻繁な書換えを起こし易い大きさであるか否かは考慮していなかった。

〔発明が解決しようとする問題点〕

データのアクセスを高速に行うため、メモリはいくつかのバンクに分割されている。それぞれのバンクは並列にアクセス可能なので、データが並列アクセスできるよう

にメモリ上に配置されていれば高速処理できるが、アクセスが一つのバンクに集中すると、一つの要素に対するアクセスが終るまで次の要素に対するアクセスはできないので処理スピードが極端に低下する。これをバンクコンフリクトという。

上記バンクコンフリクトは、配列要素を等間隔でアスセスする場合に発生し易い。FRORTRANについて言えば、配列は列方向にメモリに割付けられるので、例えば、１バンクで１要素を読出せる電子計算機において、２次元配列の行数、すなわち、一列当りの要素数を、バンク数の整数倍にして配列を行方向に処理すると、アクセスは一つのバンクに集中してしまう。

ところが、FORTRANプログラムの配列処理は配列を行方向あるいは列方向等、ある次元に沿って行うことが多い。コンパイラは配列のメモリ割付けを配列記述テーブルに従って行うが、このテーブルは原始プログラムで配列した通りの値を使って作成されるので、利用者はバンクコンフリクトを避けるために配列を行方向に参照しないようにアルゴリズムを変更するか、あるいは、バンク数を意識して次元の寸法がバンク数の整数倍にならないように配列宣言する等、プログラム作成上の工夫をしなければならなかった。

また、バッファを持つ電子計算機においても同様の問題がある。すなわち、上記バッファはいくつかのブロックに分割され、ブロックとは行と列とに並べられている。メモリも同じ大きさのブロックに分割されており、データをアクセスすると、まず、当該データの含まれるブロックがメモリからバッファに転送され、次に、必要なデータのみがバッファから演算処理装置へ転送される。

メモリ上のブロックは、それぞれバッファのどの行に転送されるかが予め決まっているため、同じ行に転送されるデータばかりアクセスすると、バッファの列数が限られているため、バッファの書換えが頻繁に発生し処理スピードが低下する。これも、前記バンクコンフリクトと同様に、配列要素を等間隔でアクセスする場合に起こり易い問題である。

本発明は上記事情に鑑みてなされたもので、その目的とするところは、従来の多次元配列のメモリ割付け方式における上述の如き問題を解消し、配列をある次元に沿って処理する場合に起こるバンクコンフリクトやバッファの頻繁な書換えを、コンパイラ側で回避し、利用者に意識させることなく多次元配列のメモリ割付けを行うことが可能な、多次元配列のメモリ割付け方式を提供することにある。

〔問題点を解決するための手段〕

本発明の上記目的は、複数のバンクから構成されるメモリに多次元配列の要素を順次静的に割付けるコンパイラにおいて、各バンクへの割付け方向に沿った要素の数がバンク数の整数倍である場合に、前記方向の要素に対して仮想的要素を追加し、その方向と直交する方向に沿った要素を複数のバンクに分散して割付けることを特徴とする多次元配列のメモリ割付け方式によって達成される。

〔作用〕

本発明においては、コンパイラが配列記述テーブルを作成するときに、次元の寸法がバンクコンクリフトやバッファの頻繁な書換えを起こし易いか否かを判定し、もし、そうであれば、寸法を適当に増やし、それに基づいて配列長，要素数等を計算し、配列記述テーブルに設定するようにしている。この配列記述テーブルに従って配列を割付けることにより、特別なハードウェアを設けることなく、コンパイラ側で柔軟にバンクコンフリクトを回避しバッファの頻繁な書換えを防止することができる。

より具体的には、配列の次元の寸法がバンクコンクリフトを起こし易い大きさのとき、例えば、バンク数の整数倍のとき、配列のある行の要素はすべて一つのバンクに存在する。このとき、配列の寸法を１増やしてメモリに割付ければ、要素の存在するバンクが１つずつずれ、行方向にアクセスしてもバンクコンフリクトが発生しなくなる。

バッファについても、同様に、配列の行方向のアクセスがある行のブロックに集中するときは、配列の寸法を増やしてメモリに割付けることにより、その行の要素の存在するブロックをずらすことができるので、バッファの書換えを減らすことができる。

〔実施例〕

以下、本発明の実施例を図面に基づいて詳細に説明する。

第２図は本発明が適用されるコンパイラの一構成例を示すものである。本コンパイラ２は、原始プログラム１を入力して、目的プログラム３を生成する機能を有するものである。

上記コンパイラ２は、原始プログラム１を入力して中間語を生成し、中間語ファイル４に出力する原始プログラム解析部5,上記中間語に対して実行効率が良くなるよう最適化を行う中間語最適化部6,目的プログラムの実行に必要な領域の割付けを行うメモリ割付け部7,上記中間語の命令列に対してレジスタの割当てを行うレジスタ割当て部8,上記中間語を機械語に変換して出力する目的プログラム出力部９から構成されている。更に、上記原始プログラム解析部５は、字句解析部10,構文解析部11,意味解析部12,辞書作成部13および中間語作成部14から構成されている。

本発明は、原始プログラム解析部５中の辞書作成部13、および、該辞書作成部13が作成した配列記述テーブルに基づき、配列にメモリを割付けるメモリ割付け部７に適用されるものである。

まず、第一の実施例として、１バンクで４バイトを読出せるバンク数16のメモリを持つ電子計算機システムにお

いて、第３図に示す如きFORTRANの原始プログラムが与えられたときに辞書作成部13が作成する、前記配列記述テーブルを含むテーブル類について第４図を用いて説明する。

前記辞書作成部13に入る前に字句解析部10,構文解析部11,意味解析部12を経て、第３図の原始プログラムはＡの属性を宣言する文で、具体的には実数型４バイトの２次元配列で、その寸法は16×16であることがわかっている。

辞書作成部13は、記号情報テーブル15,記号名テーブル16,配列記述テーブル17を作成する。記号情報テーブル15には、記号名アドレス15a,記号の型15b,その記号が「配列」であることを示すフラグ15c,該フラグがONのとき配列記述テーブルアドレス15dが入る。本実施例では、記号の型15bは実数型４バイト、フラグ15cはONである。記号名アドレス15aでポイントされた記号名テーブル16には記号名が入り、本実施例では「Ａ」である。

配列記述テーブルアドレス15dでポイントされた配列記述テーブル17には、その配列の次元の数17a,配列長17b,要素数17c,第１次元の寸法17d,第１次元のマルチプライヤ17e,第２次元の寸法17f,第２次元のマルチプライヤ17gが入る。次元の寸法とマルチプライヤは、次元の数だけ繰り返し配列記述テーブル17に入れられる。ここで、マルチプライヤとは、その次元の添字を１増やしたとき、メモリ上で物理的に何要素離れているかを示す値である。

配列記述テーブル17には、まず、配列宣言文の解析結果から直に分かる次元の数17a,第１次元の寸法17d,第２次元の寸法17fを設定し、次に配列情報計算により、配列長17b,要素数17c,各マルチプライヤ17e,17gを計算して配列記述テーブル17を完成させる。なお、第４図の各欄のコロン（：）の後は、実施例での具体的な値を示しており、＊は配列情報計算後に設定された値を示している。

上記配列情報計算を第１図（ａ）のフローチャートを用いて説明する。ここで、メモリのバンク数は、例えば、コンパイラオプションで指定された電子計算機の構成に関する情報から既に分かっているものとする。

まず、ステップ21で次元の寸法がバンク数の整数倍か否かを判定し、整数倍であればステップ22に進み、（次元の寸法＋１）の値で配列記述テーブルの次元の寸法のフィールドを書換え、ステップ23に進む。なお、次元の寸法がバンク数の整数倍でなければ、何もしないでステップ23に進む。

ステップ23では、マルチプライヤを計算し、配列記述テーブルの該当するフィールドに値を設定する。上記ステップ21～23を、（次元数の－１）回繰り返した後、ステップ24で配列長，要素数，最終次元のマルチプライヤを計算し、配列記述テーブルに値を設定してテーブルを完成させる。

本実施例について具体的に説明すれば、まず、ステップ21で第１次元の寸法について判定を行うが、これが16，メモリのバンク数も16なので、ステップ22の処理を行い、配列記述テーブル17の第１次元の寸法17bを「17」に書換える。次に、ステップ23で、マルチプライヤを計算し、エリア17eに「１」を設定する。次元の数が２なので、繰り返しはなく、ステップ24の処理を行う。要素数は、（第１次元の寸法×第２次元の寸法）で計算されるので、17×16＝272を17cに設定する。

配列長は（要素数×１要素のバイト数）で計算されるので、272×４＝1088を17bに設定する。

第２次元のマルチプライヤは第１次元の寸法と等しいので、17を17gに設定する。これで、配列テーブル17が完成する。

第５図は第４図の配列記述テーブル17に従って前述の配列「Ａ」をメモリ18に割付けた状態を示している。ここで、ai,jをＡの第ｉ行第ｊ列の要素とすれば、第次元の寸法（１列当りの要素数）が17なので、Ａの第２列の先頭要素$a_{1,2}$は、メモリ上ではバンク２の２番目に位置付けられる。

行方向の処理についてＡの第１行を例として調べてみれば、アクセスする要素$a_{1,1}$、$a_{1,2}$、$a_{1,3}$、‥‥、$a_{1,16}$（○で囲まれている）の存在するバンクは、それぞれ、バンク1,バンク2,バンク3,‥‥，バンク16とすべて異なるため、並列アクセスが可能である。従ってＡを行方向に処置しても、バンクコンフリクトは発生しない。

また第５図のように、要素$a_{1,1}$がメモリの１ワードを占有し、各列ベクトルの最後に仮想的要素を１個追加する場合、要素$a_{i,j}$のメモリアドレス（ワード）は、（ｉ－１）×（第１次元のマルチプライヤ）＋（ｊ－１）×（第２次元のマルチプライヤ）によって求められる。この場合、（ｉ－１）×１＋（ｊ－１）×17によって計算できる。勿論、一般の要素$a_{i,j}$に対しても同じ計算式によってアドレス計算ができる。

第６図は、従来技術において、バンク数を考慮せずに、与えられた原始プログラムの通りに配列Ａをメモリ18に割付けた状態を示す図である。この場合には、Ａの第１行の要素$a_{1,1}$、$a_{1,2}$、$a_{1,3}$、‥‥、$a_{1,16}$（○で囲まれている）は、すべてバンク１に存在するため、Ａを行方向に処理しようとするとバンクコンクフリクトが発生する。

第７図は上記実施例のメモリ割付け処理の説明図である。第７図（ａ）は当初の１×16の配列Ａを示しており、第７図（ｂ）はこれを行方向に１伸ばした状態、すなわち、配列情報計算後の状態を示している。第７図（ｂ）中の＊印の仮想的要素が、第５図に＊印で示した位置に配置されていると考えて良い。

上記実施例においては、ステップ21において、「次元の寸法がバンク数の整数倍か否か」のみを判定しているが、これは、逆にバンク数が次元の寸法の整数倍になる

か否かをも含めて判定するようにしても良い。

次に、第二の実施例として、バッファを持つ電子計算機において、バッファの頻繁な書換えによる処理スピードの低下を防ぐ例を、第８図を用いて説明する。バッファ19は16個のブロック20に分割されており、ブロックは８行２列に配列されている。１ブロックは16バイトで、４バイトの要素なら４個入る。メモリ18も同じ大きさのブロックに分割されている。

第３図の原始プログラムが与えられた場合、前述の実施例と同じ手順で原始プログラムを解析した後、第１図（ｂ）に示すフローチャートに従って配列情報を計算する。

まず、ステップ31でバッファの１列に入る要素数が配列の１列当りの要素数の整数倍、または、逆に、配列の１列当りの要素数がバッファの１列に入る要素数の整数倍になっているか否かを判定する。どちらかの条件を満たせば、ステップ32に進み（次元の寸法＋１ブロックに入る要素数）で配列記述テーブルの寸法を書換え、ステップ33に進む。なお、どちらの条件も満たさなければ、直接ステップ33に進み、マルチプライヤを計算して配列記述テーブルに値を設定する。

これを（次元の数－１）回繰り返し、最後にステップ34で配列長，要素数，最終次元のマルチプライヤを計算し、配列記述テーブルに値を設定する。なお、バッファの１列に入る要素数は、コンパイラオプション等で既に分かっているものとする。

本実施例では、バッファの１列に入る要素数が32,配列の１列当りの要素数が32なので、上記ステップ31の条件を満たし、ステップ32に進んで１ブロックに入る要素数が４なので配列記述テーブルの配列の寸法を20に書換える。最終的に作成された配列記述テーブルは第９図のようになる。

前にも使用した第８図は、第９図の配列記述テーブルに従って配列Ａを割付けした状態と、配列Ａの第１行をバッファ19に転送した状態とを示している。Ａの第１行の要素$a_{1,1}$、$a_{1,2}$、$a_{1,3}$、‥‥、$a_{1,16}$（○で囲まれている）を含むブロックは、バッファ19の同一行には２個ずつしか対応しないので、バッファの書換えを行わないで16個のブロックをバッファ19に転送することが可能である。

これに対して、第10図は従来の方法で配列Ａをメモリ18に割付けた状態と、バッファ19とを示す図である。この場合には、Ａの第１行の要素を含むブロックは、バッファ19の第１行に８個、第５行に８個対応しているため、最初の２個ずつを転送した後はバッファを書換えながら転送しなければならず、処理性能が低下する。

上記二つの実施例によれば、利用者が配列を処理するプログラムを作成する際、バンクコンフリクトやバッファの頻繁な書換えを避けるためにアルゴリズを工夫したり、メモリのバンク数やバッファの１列に入る要素数を意識して配列宣言したりする必要がなくなり、この種のプログラム作成上の煩雑な配慮をしなくても、安定した配列アクセス性能を得ることができる。

上記実施例では、バッファの１ブロックに入る要素数だけ配列の寸法を増やしたが、メモリの増加が気になる場合は増やす量を１ブロックに入る要素数より少なくしても、何要素が後にはブロックがずれてくるため、効果が得られる。

なお、上記二つの実施例では、配列の寸法の宣言が定数の場合を示したが、本発明はこれに限定されるべきものではなく、宣言が変数でプログラムのどこかでその値が明示的に与えられる場合，宣言が演算式の場合にも適用可能なものである。

また、二つの実施例はFORTRAN言語による例を示したが、本発明は、配列をメモリに静的に割付けるプログラミング言語全般に広く適用可能であることは言うまでもない。

さらに、二つの実施例では、配列の要素を列方向に順次割付ける場合を取り上げたが、勿論、配列の要素を行方向に順次割付ける場合についても同様にして、割付け方向と直交する方向に沿う要素が一つのバンクに集中しないように分散して割付けることができる。

〔発明の効果〕

以上述べた如く、本発明によれば、多次元配列を静的にメモリに割付けるコンパイラおいて、配列の次元の寸法がバンクコンフリクトやバッファの頻繁な書換えを起こし易い大きさである場合に、前記次元の寸法を増やして前記バンクコンフリクトやバッファの頻繁な書換えが発生し難い大きさにして、割付け情報を設定するようにしたので、配列をある次元に沿って処置する場合に起こるバンクコフリクトやバッファの頻繁な書換えを、コンパイラ側で柔軟に回避し、利用者に意識させることなく多次元配列のメモリ割付けを行うことが可能な、多次元配列のメモリ割付け方式を実現できるという顕著な効果を奏するものである。

【図面の簡単な説明】

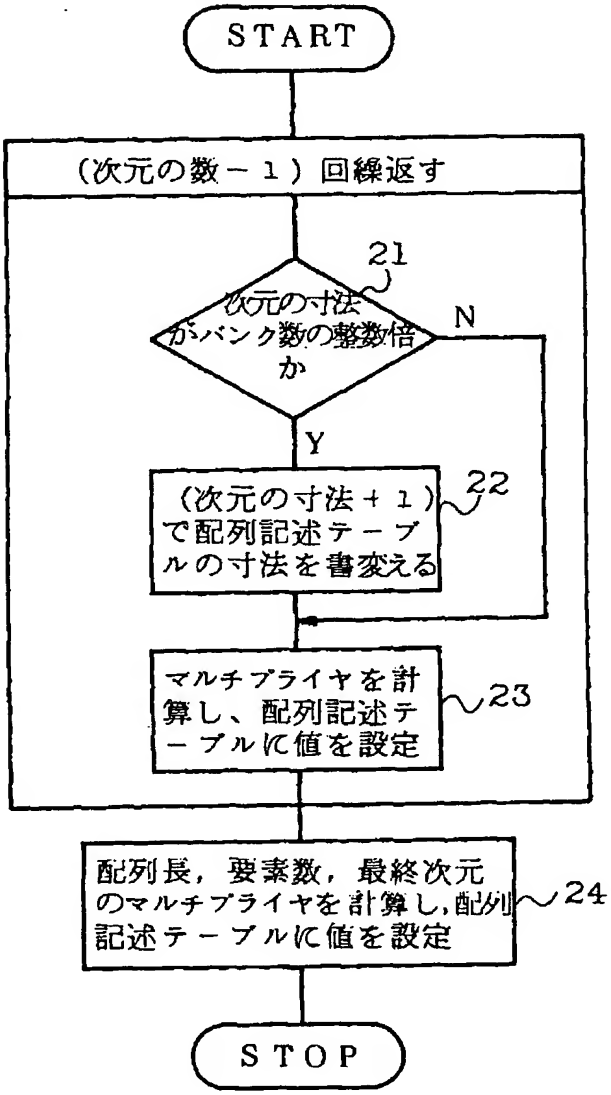
第１図（ａ）および（ｂ）は配列情報計算の処理の流れを説明するフローチャート、第２図は本発明が適用されるコンパイラの一構成例を示す図、第３図は辞書作成部の動作を説明するための原始プログラムの例を示す図、第４図および第９図は辞書作成部が作成するテーブルの説明図、第５図および第８図は本発明の実施例によるメモリ割付け例を示す図、第６図および第10図は従来の方法によるメモリ割付け例を示す図、第７図は実施例の動作原理の説明図である。

1:原始プログラム、2:コンパイラ、3:目的プログラム、4:中間語ファイル、5:原始プログラム解析部、6:中間語最適化部、7:メモリ割付け部、8:レジスタ割当て部、9:目的プログラム出力部、10:字句解析部、11:構文解析部、12:意味解析部、13:辞書作成部、14:中間語作成

9　　10

部、15:記号情報テーブル、16……記号名テーブル、17:　　＊ック。
配列記述テーブル、18:メモリ、19:バッファ、20:ブロ ＊

【第1図（a）】



【第9図】

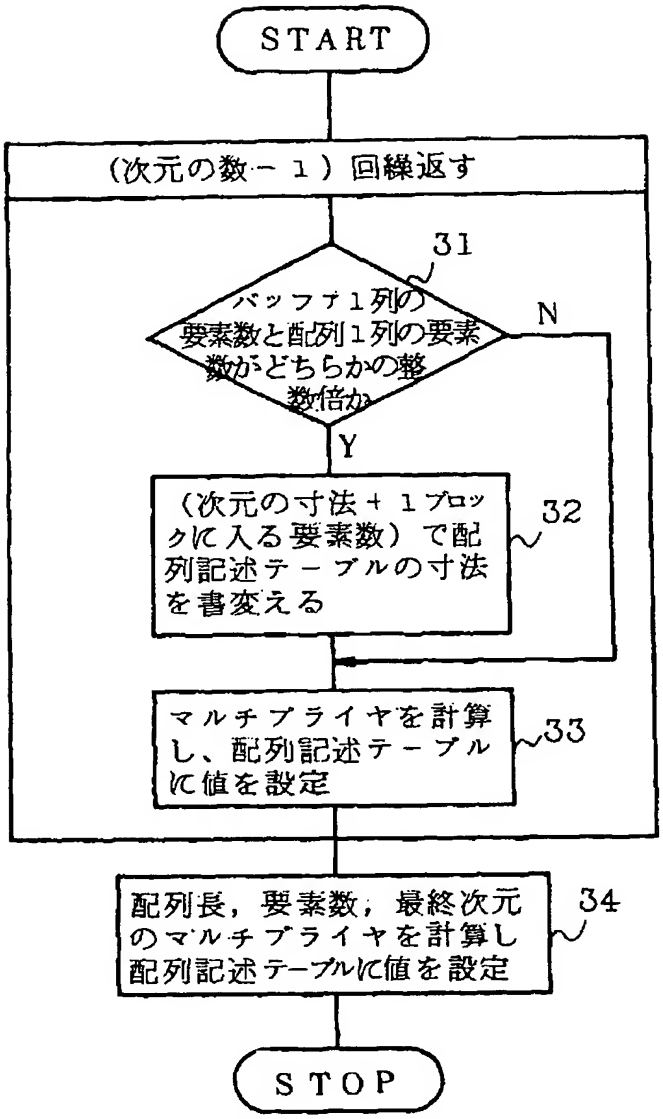
17

次元の数：	2	17-1
配列長：	1280*	17-2
要素数：	320*	17-3
第1次元の寸法：	16→20*	17-4
第1次元のマルチプライヤ：	1*	17-5
第2次元の寸法：	16	17-6
第2次元のマルチプライヤ：	20*	17-7

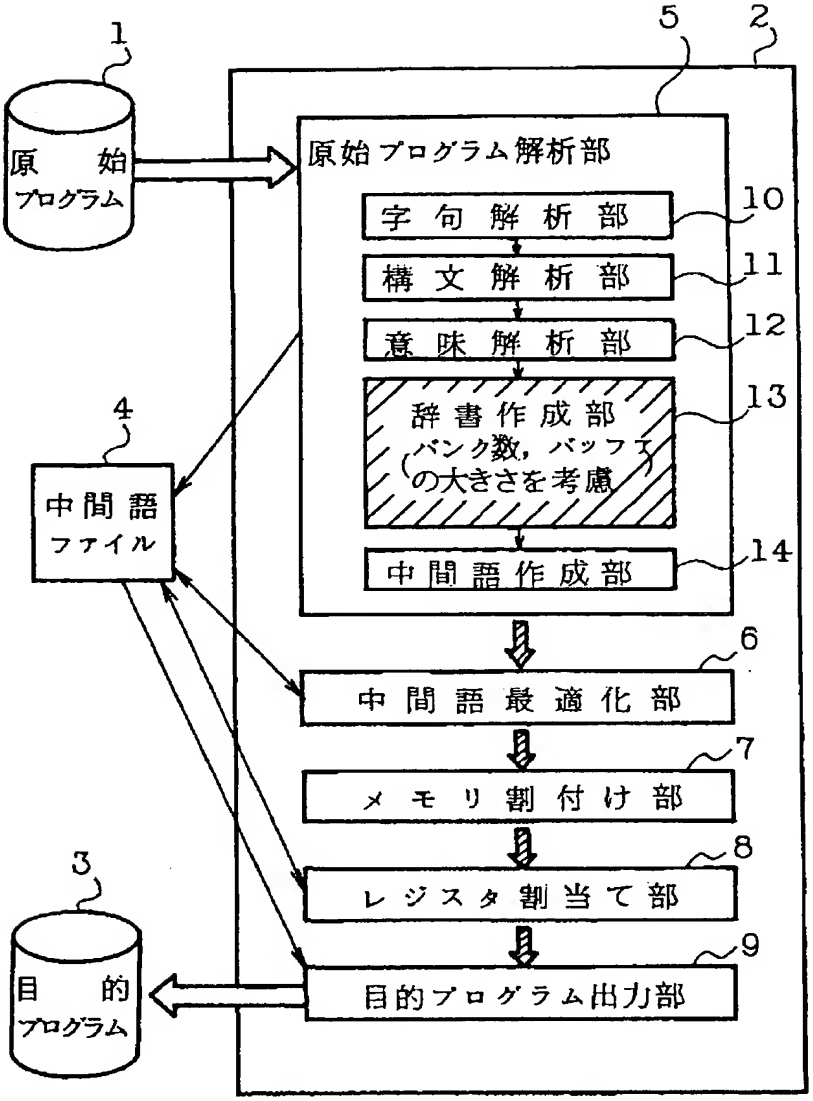
【第3図】

```
REAL * 4    A
DIMENSION   A (16, 16)
    :
```

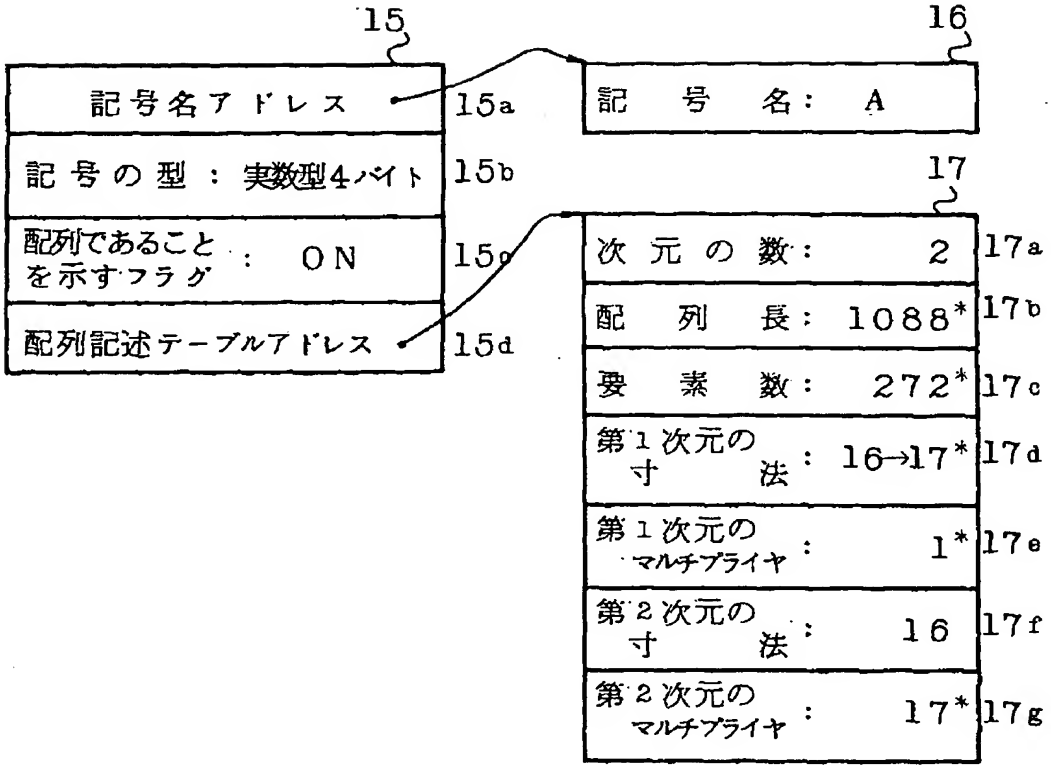
【第1図（b）】


START
（次元の数－１）回繰返す
31
バッファ１列の要素数と配列１列の要素数がどちらかの整数倍か
N
Y
（次元の寸法＋１ブロックに入る要素数）で配列記述テーブルの寸法を書変える
32
マルチプライヤを計算し、配列記述テーブルに値を設定
33
配列長，要素数，最終次元のマルチプライヤを計算し配列記述テーブルに値を設定
34
STOP

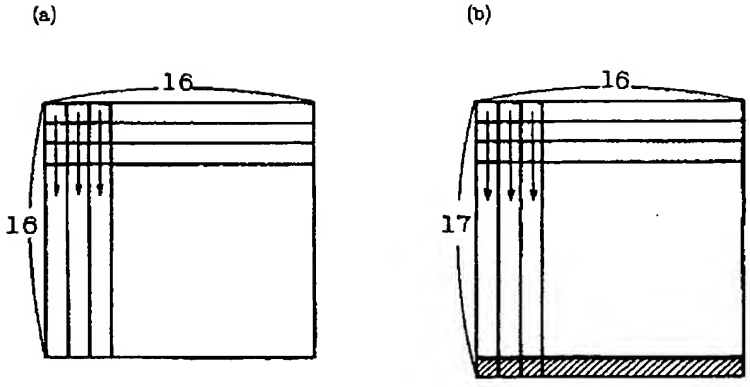
【第2図】


1
原始プログラム
5
2
原始プログラム解析部
字句解析部
10
構文解析部
11
意味解析部
12
辞書作成部
(バンク数，バッファの大きさを考慮)
13
中間語作成部
14
4
中間語ファイル
6
中間語最適化部
7
メモリ割付け部
8
レジスタ割当て部
9
目的プログラム出力部
3
目的プログラム

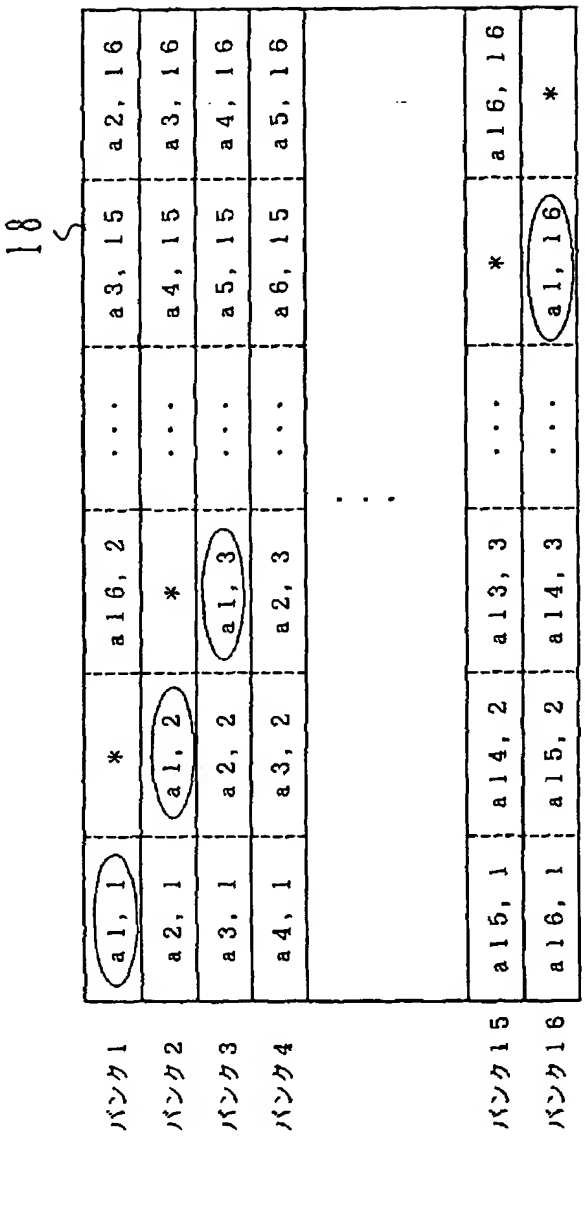
【第4図】


15
16
記号名アドレス
15a
記号の型：実数型4バイト
15b
配列であることを示すフラグ：ON
15c
配列記述テーブルアドレス
15d
記号名：A
17
次元の数：2
17a
配列長：1088*
17b
要素数：272*
17c
第1次元の寸法：16→17*
17d
第1次元のマルチプライヤ：1*
17e
第2次元の寸法：16
17f
第2次元のマルチプライヤ：17*
17g

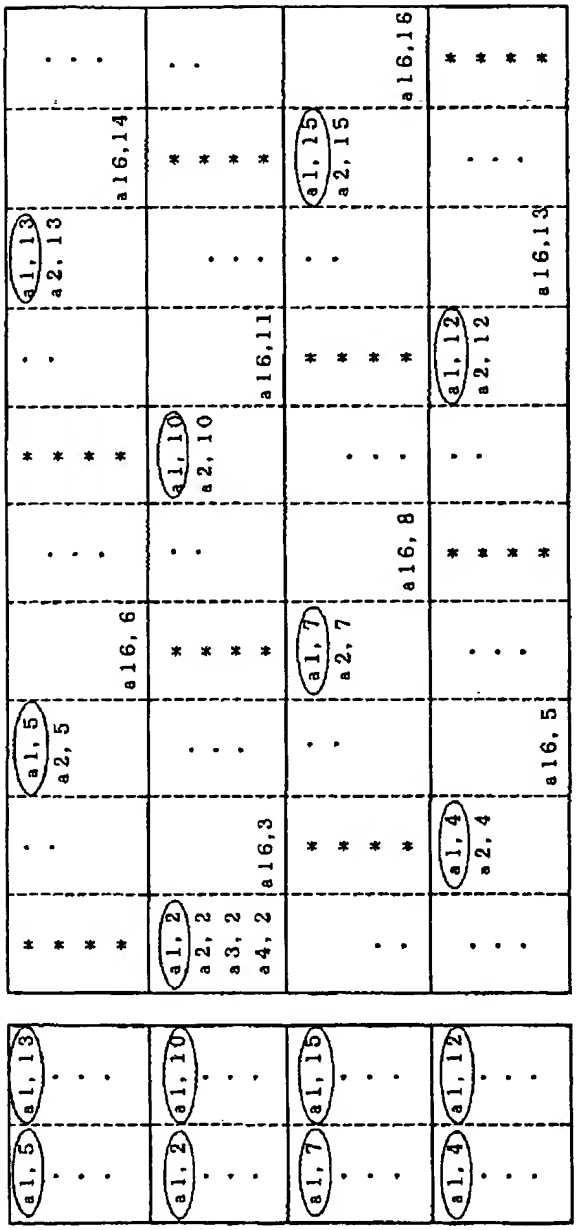
【第7図】


(a)
16
16
(b)
16
17

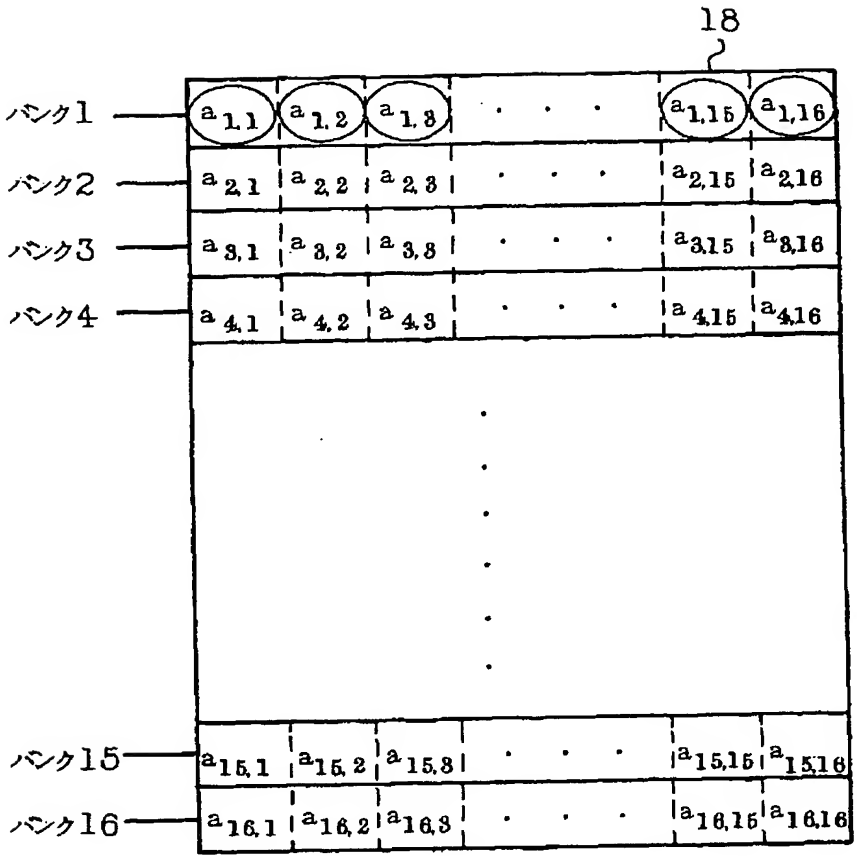
【第5図】


18
バンク1
バンク2
バンク3
バンク4
バンク15
バンク16
a1, 1
a2, 1
a3, 1
a4, 1
a15, 1
a16, 1
*
a1, 2
a2, 2
a3, 2
a14, 2
a15, 2
a16, 2
*
a1, 3
a2, 3
a13, 3
a14, 3
a3, 15
a4, 15
a5, 15
a6, 15
*
a1, 16
a2, 16
a3, 16
a4, 16
a5, 16
a16, 16
*

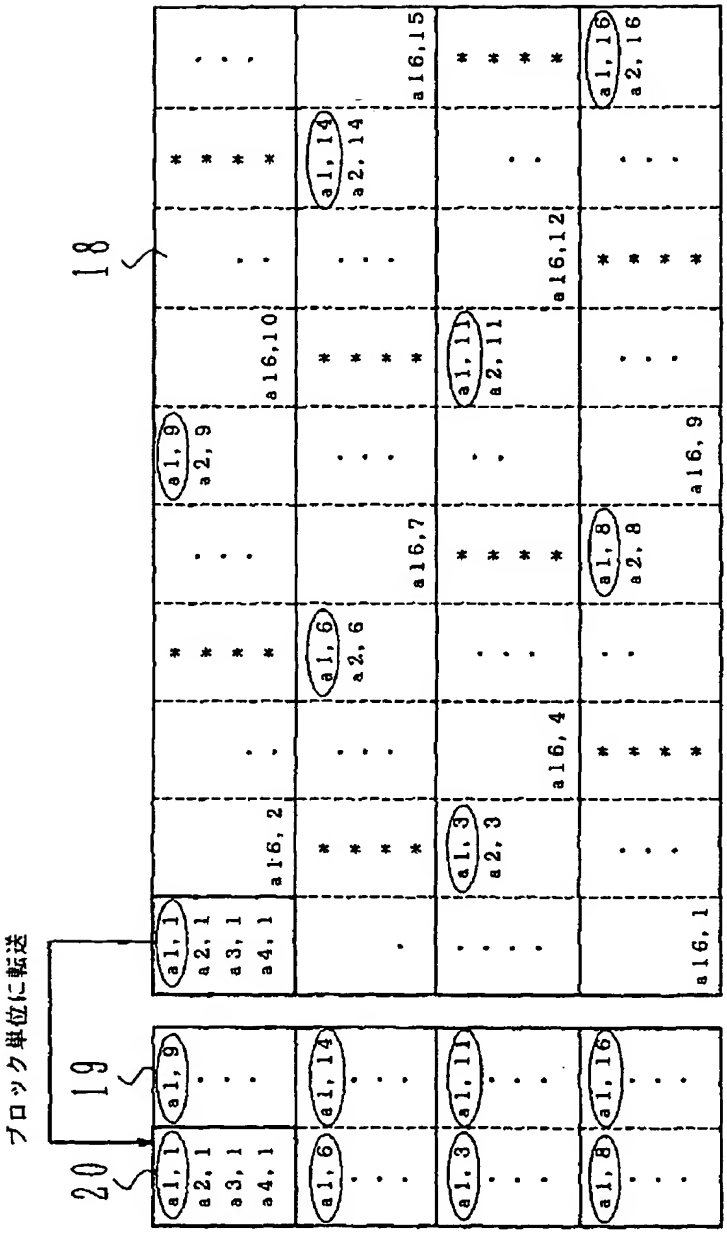
【第8図（その2）】


a1, 5
a1, 13
a1, 2
a1, 10
a1, 7
a1, 15
a1, 4
a1, 12

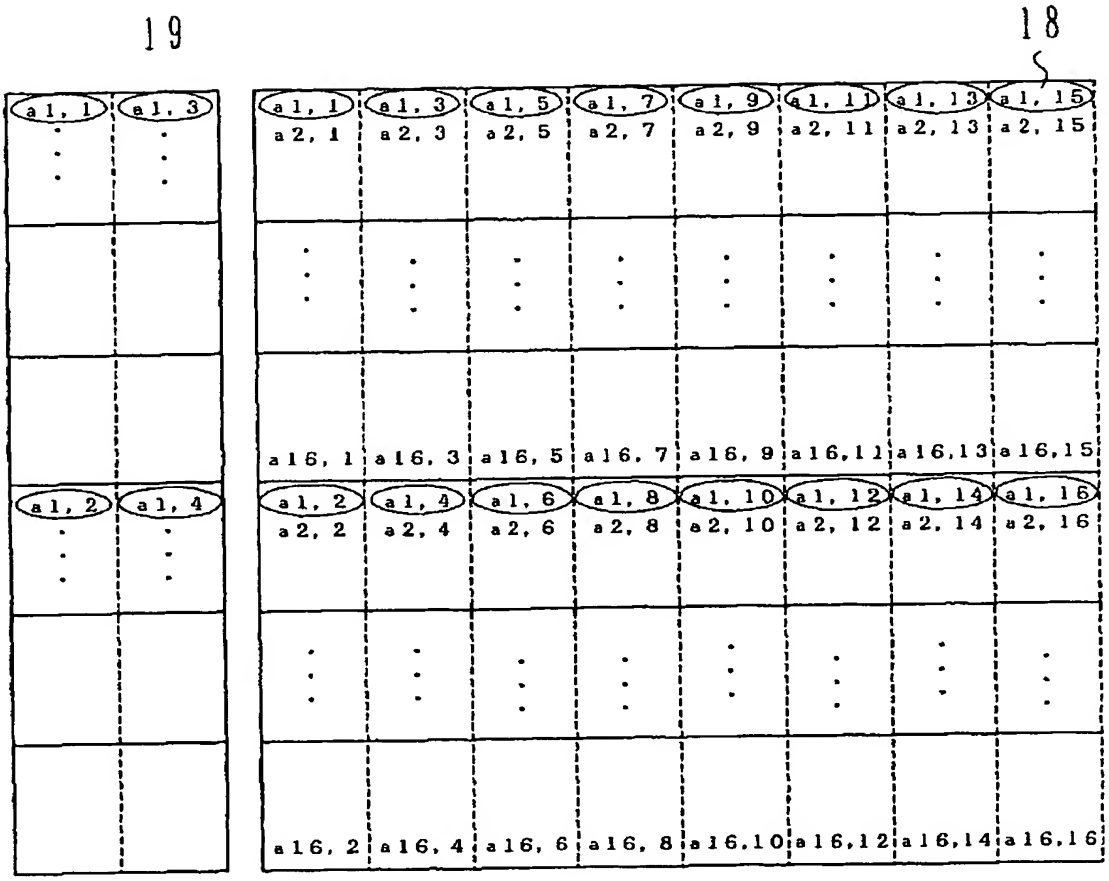
【第6図】


18
バンク1
バンク2
バンク3
バンク4
バンク15
バンク16
a1,1
a1,2
a1,3
a1,15
a1,16
a2,1
a2,2
a2,3
a2,15
a2,16
a3,1
a3,2
a3,3
a3,15
a3,16
a4,1
a4,2
a4,3
a4,15
a4,16
a15,1
a15,2
a15,3
a15,15
a15,16
a16,1
a16,2
a16,3
a16,15
a16,16

【第8図（その1）】


18
19
20
ブロック単位に転送
a1, 1
a2, 1
a3, 1
a4, 1
a16, 2
a16, 4
a16, 1
a1, 3
a2, 3
a1, 6
a2, 6
a16, 7
a1, 8
a2, 8
a16, 9
a1, 9
a2, 9
a16, 10
a1, 11
a2, 11
a16, 12
a1, 13
a2, 14
a16, 15
a1, 16
a2, 16

【第10図】


19
18
a1, 1
a1, 3
a1, 2
a1, 4
a1, 1 a1, 3 a1, 5 a1, 7 a1, 9 a1, 11 a1, 13 a1, 15
a2, 1 a2, 3 a2, 5 a2, 7 a2, 9 a2, 11 a2, 13 a2, 15
a16, 1 a16, 3 a16, 5 a16, 7 a16, 9 a16, 11 a16, 13 a16, 15
a1, 2 a1, 4 a1, 6 a1, 8 a1, 10 a1, 12 a1, 14 a1, 16
a2, 2 a2, 4 a2, 6 a2, 8 a2, 10 a2, 12 a2, 14 a2, 16
a16, 2 a16, 4 a16, 6 a16, 8 a16, 10 a16, 12 a16, 14 a16, 16